

Grid Data Mirroring Package

Grid Data Mirroring Package*(GDMP)

User Guide for GDMP 2.0

Heinz Stockinger 1), Asad Samar 2), Shahzad Muzaffar 3),
Flavia Donno 4), Andrea Domenici 5)

1) CERN, European Organization for Nuclear Research, Geneva, Switzerland, Heinz.Stockinger@cern.ch

2) California Institute of Technology, Pasadena, California, Asad.Samar@cern.ch

3) Fermi National Laboratory, Batavia, Illinois, muzaffar@fnal.gov

4) INFN Pisa, Italy, Flavia.Donno@pi.infn.it

5) INFN & University of Pisa, Italy, andrea@sssup.it

October 24, 2001

Abstract

The GDMP client-server software system is a generic file replication tool that replicates files securely and efficiently from one site to another in a Data Grid environment using several Globus Grid tools. In addition, it manages replica catalogue entries for file replicas and thus maintains a consistent view of names and locations of replicated files. Files of arbitrary file format can be replicated. For Objectivity database files a particular plug-in exists. All files are assumed to be read-only.

GDMP is a collaboration between the European DataGrid [2] project (in particular the Data Management work package, Work Package 2 (WP2) [3]) and Particle Physics Data Grid (PPDG) [9]. GDMP version 2.0 is part of the official DataGrid software system.

This user guide gives detailed instructions for installation and usage of GDMP. In addition, a generic replica catalogue API in C++ for the Globus Replica Catalogue is provided and described.

*previously called Grid Data Management Pilot

Contents

1	Introduction	3
2	New Features and Improvements in Version 2.0	3
3	Software Installation	4
3.1	Required Software	4
3.2	Installation Instructions	5
3.3	RPM Installation	7
3.4	Configuration Instructions	7
3.5	The GDMP Directories	10
3.6	GDMP Server Certificate	10
3.6.1	CERN Certificate	10
3.7	Globus Certificate	10
4	Data Replication and Background for GDMP Usage	10
4.1	Directories and File Replication	11
4.2	Filenames	11
4.3	Globus Replica Catalogue	12
4.4	Internal File Catalogues	13
5	Using GDMP	13
5.1	Quick Start Guide to run GDMP	13
5.2	The Mechanics of GDMP	15
5.3	Security Issues for GDMP Server and Clients	17
5.4	Subscription to Remote Servers	17
5.5	Notification	17
5.6	System States for File Replication Process	18
5.7	Replication Policy	19
5.8	Network Failures	19
5.9	Some Program Restrictions	19
6	GDMP Tools	19
7	Support for a Mass Storage System	25
7.1	Motivation	25
7.2	Flow of Control	26
7.3	The Interface	26
7.4	Staging States	27
7.5	Interface to HRM	27
8	Appendix A: Configuring GDMP with inetd	28
8.1	Inetd Background	28
8.2	Configuration Steps done by GDMP Installation Program	28
8.3	Example Configuration for inetd	29
9	Appendix B: Example for gdmf.conf	30
10	Appendix C: Usage of Grid Security Infrastructure	35

11 Appendix D: Replica Catalogue API	36
11.1 Description of the Programming Interface	36
11.2 Usage Instructions	36
12 Appendix E: BrokerInfo API	38
12.1 Description of the Programming Interface	38
12.2 Usage Instructions	39

1 Introduction

The GDMP [10, 6, 11] software tools provide automatic, asynchronous replication (mirroring) of arbitrary files of any data format (called “flat files” in this document) and Objectivity database files in a Data Grid environment. In principle, a site, where files are created, has to notify the GDMP software which in turn notifies all the other sites in the Grid about the new files. A file is ready for mirroring only when it is guaranteed that it is closed and no other process will write into it anymore. It is the responsibility of the source site to determine when a file is ready for transfer. The destination sites receive a file listing of all the new files available at the source site and can determine themselves when to start the actual data transfer. The data transfer is done with GridFTP clients and GridFTP servers (i.e. GSI enabled and modified wu-ftp server). Replicas can be registered in a replica catalogue (based on the Globus replica catalogue using an LDAP [13] implementation) and thus made available to the Grid. A user has to be authenticated and authorised before contacting any remote site. Authentication and authorisation are based on the Grid Security Infrastructure (GSI security) layer available from Globus.

WP2 (the Data Management work package in the European DataGrid) is working on a generic replica catalogue API. The GDMP code distribution contains a preliminary C++ API (see Appendix C in Section 11) for a central Globus replica catalogue based on LDAP. Note that GDMP internally also uses the Globus replica catalogue for management of replica information.

To sum up, the software package contains three major parts:

- GDMP for file replication. The main purpose of this User Guide is to explain its functionality.
- a generic Replica Catalogue API in C++
- BrokerInfo: this is an API to handle information coming from Scheduling system of WP1 (Workload Management, DataGrid) which is provided to Data Grid application running on the first DataGrid testbed but is not required for GDMP itself.

2 New Features and Improvements in Version 2.0

The following is a list of new features in GDMP version 2.0 and changes from version 2.0alpha:

1. GDMP depends on Globus Toolkit 2.0 Alpha Release 9 or higher.
2. The GridFTP client library is used instead of NC-FTP.
3. A new tool `gdmp_cert_update` allows to switch from a CERN to a Globus certificate for the GDMP server
4. in `gdmp.conf`, three new variables have been added: `GLOBUS_LOCATION`, `OBJECTIVITY_DIR`, `ORBACUS_DIR`

New Features in GDMP 2.0alpha (30 Sep. 2001)

The following is a list of new features in GDMP version 2.0alpha and changes from version 1.2.2:

1. The Globus replica catalogue is used for storing file replica information about logical and physical filenames as well as file attributes (size, timestamp, CRC checksum) and physical file locations.
2. An additional notification system for file transfers is added.
3. Most of the command line tools have new short, single letter option names instead of entire words, e.g. `gdmp_ping -r` instead of `gdmp_ping -remotehost`.
4. The tool previously called `gdmp_cleanup` has been renamed to `gdmp_catalogue_cleanup`.
5. A new tool `gdmp_register_local_file` has been added that needs to be called *before* `gdmp_publish_catalogue`.
6. A local file catalogue keeps track of all files GDMP manages locally.
7. The tool previously called `gdmp_replicate_file_get` has been renamed to `gdmp_replicate_get`.
8. The tool `gdmp_setup` is no longer available as a self-contained installation and configuration procedure is now used.
9. Instead of environment variables, a GDMP configuration file (`gdmp.conf`) is used.
10. The staging scripts called through `GDMP_STAGE_TO_MSS` and `GDMP_STAGE_FROM_MSS` (Section 7) now require two arguments instead of one.
11. A generic replica catalogue API in C++ for the Globus Replica Catalogue is provided.
12. An appendix on the practical usage of the Grid Security Infrastructure has been added.

3 Software Installation

In the following section we give detailed instructions for the installation of the software package. We provide a source code distribution as well as a binary distribution and explain both installation procedures.

3.1 Required Software

The GDMP software runs and has been tested on **Linux RedHat 6.1 and 6.2** on top of Globus Toolkit 2.0 Alpha Release 9. The GDMP software consists of several executables and a server named `gdmp_server` which runs using the Internet daemon (`inetd`) (see Section 8 for more details) at the host that produces files. The host has to be reachable by the “outside world” and cannot be behind the local firewall since permanent network connections to this machine are required. The same is true for the FTP server.

A site has to have the following software installed locally:

- GDMP software version 2.0
- Globus Toolkit 2.0 Alpha 9 (special release for European DataGrid) or higher (including the WU-FTP server),
<http://datagrid.in2p3.fr/distribution/globus/alpha-9/>
- g++ compiler `gcc-2.91.66` or `gcc-2.95.2`

- GNU Make version 3.77 or higher
- GNU Autoconf version 2.13
- GNU libtool 1.4
- GNU automake 1.4-p2
- GNU m4 1.4
- RPMv3 or higher

GDMP uses file locks and they have to be enabled on the system where GDMP is installed.

In case Objectivity files are replicated, Objectivity/DB Version 5.x or 6.x is required. Furthermore, the Objectivity bootfile as well as all the Objectivity database files have to be reachable by the FTP and the GDMP server. This guarantees a continuous data transfer from the local to the remote disk via FTP. File access via the Objectivity AMS is not supported through GDMP.

If the BrokerInfo library is built/used, also the Condor ClassAd v2.6 software must be installed and available to users.

Note that for a binary distribution the GNU Autoconf and lib tools are not required.

3.2 Installation Instructions

The following instructions apply to the source code distribution. Before starting the compilation, the following environment variables can be set or configured with configure (see below):

GLOBUS_LOCATION: base directory of the Globus installation

OBJY_DIR: base directory for Objectivity installation - only required if you plan to build GDMP with Objectivity support.

CLASSAD_DIR: If you plan to build the BrokerInfo library (not required for GDMP nor the Replica Catalogue, the environment variable **CLASSAD_DIR** has to point to the Condor ClassAd installation directory.

After unpacking the GDMP source distribution tar file, or getting the code directly from the CVS repository, change your working directory to be the GDMP base directory and run the following command:

```
./bootstrap
```

At this point you are ready to run the **configure** command. The configure command should be invoked as follows:

```
./configure [option] where option can be the following:
```

```
--help
```

```
--prefix=<installation dir> it is used to specify the GDMP installation dir. The default installation dir is /opt/edg.
```

--enable-brokerinfo it is used to enable the build of the BrokerInfo User API library. Note that the BrokerInfo API is provided by the Workload management work package and is *not* required for GDMP. By default this option is turned off. If the environment variable **CLASSAD_DIR** is not set, you can specify the ClassAd installation directory using the option **--with-classad-install=<dir>**.

--enable-gdmp it is used to build the GDMP package and the Replica Catalogue User API library. By default this option is turned on. You can use the option **--disable-gdmp** to only build the Replica Catalogue User API library.

--with-objectivity it is used to enable Objectivity support. By default this option is turned on. If the environment variable **OBJY_DIR** is not set, you can specify the ClassAd installation directory using the option **--with-objectivity-install=<dir>**.

--with-globus-install=<dir> allows to specify the Globus install directory without setting the environment variable

--with-globus-flavor=<dir> allows to specify a specific Globus flavour. Possible values are **gcc32dbgpthr**, the default, and **gcc32dbg**.

During the configure step, a spec file (**gdmp[-objy].spec**) will be produced in the GDMP source directory to produce a flavour specific version with or without Objectivity.

gmake

Run **gmake** in the GDMP source code directory. Be careful to use the GNU distribution of make. Proprietary versions do not work all the time. (GDMP, RC, BrokerInfo have only been built successfully on RedHat6.1 and RedHat6.2).

gmake install

In order to install the package in the installation directory specified by the **--prefix** option during the "configure" step, you can now issue the command **gmake install** in the GDMP source tree.

gmake -i dist

The command **gmake -i dist** will produce in the GDMP source directory a binary gzipped tar ball of the GDMP distribution. This tar ball can be unwinded on a different machine. This step is not required for each installation. This tar ball can be used as source for the RPM creation.

rpm -ba gdmp[-objy].spec

In order to create an RPM for GDMP 2.0, take the tar ball created during the previous step and copy it into the rpm **SOURCES** directory, usually located in **/usr/src/redhat/SOURCES**. Copy the generated spec file (**gdmp[-objy].spec**) into the rpm **SPECS** directory, usually located in **/usr/src/redhat/SPECS**. Make sure the **PATH** for root is set in such a way that the GNU autotools, gmake and the compiler can be used. Execute the command above.

gdmp/utils/configure_gdmp <gdmp-install-dir> <userid> <port>

The command `gdmp/utils/configure_gdmp` is executed as *root* and will properly configure the `gdmp_server` in the system files. The script requires three input parameters: `GDMP_INSTALL_DIR`, `userid` and `port`. For further details refer to the Section 3.4. The `configure_gdmp` script does not set any user path. It is responsibility of the system administrator of the machine to add, if necessary, the directories `GDMP_INSTALL_DIR/utils` and `GDMP_INSTALL_DIR/bin` to the default user path.

3.3 RPM Installation

In order to install the GDMP RPM with a given flavour (Objectivity or not) execute the following command as root:

```
rpm -ivh [--prefix <installdir>] gdmp[-objy]-2.0-1.i386.rpm
```

By default the rpm installs the software in the `/opt/edg/gdmp` directory. Using the `--prefix` directive, you can relocate the software and install it under a different directory.

If you want to install the rpm as a non-root user, you should have a private copy of the RPM databases in a private directory (you can copy all `*.rpm` files from `/var/lib/rpm` in a directory where you have write access) or you should have write access to the default RPM database directory `/var/lib/rpm` and the rpm files in that directory. Then you can use the command:

```
rpm -ivh [--prefix <installdir>] [--dbpath <RPM database dir>] \
      gdmp[-objy]-2.0-1.i386.rpm
```

where `installdir` is the directory where you want to install the software and `rpm database dir` is your own private RPM database directory (you do not need to specify such a parameter if you have write access to the default `/var/lib/rpm` dir and its content).

After installing the binary RPM, the user needs to make sure that `GDMP_INSTALL_DIR/lib`¹ is included in `LD_LIBRARY_PATH`.

Furthermore, the script `configure_gdmp` can be used as root to execute the configuration root steps.

In addition to binary RPMs, also source RPMs (`gdmp[-objy]-2.0-1.src.rpm`) are available on the GDMP web page under “Software” at: <http://cmsdoc.cern.ch/cms/grid>

For further information on RPM please consult the man pages or <http://www.rpm.org>.

3.4 Configuration Instructions

Once GDMP is installed properly, the GDMP server and the client applications need to be configured. For the GDMP server, the following parameters have to be set which will be stored in a GDMP configuration file, `gdmp.conf`. All values are separated with a “=” and a sample configuration is given below. No extra characters are allowed after the values specified. Some variables are optional (O) and others are compulsory (C). Appendix B lists the contents of the entire file and gives additional comments. In the example below we assume that GDMP is installed on `host1.cern.ch` and a replica catalogue service on a second machine called `host2.cern.ch`.

¹`GDMP_INSTALL_DIR` is set in `gdmp.conf`

- `GDMP_INSTALL_DIR=/usr/local/grid/gdmp:` (C) - this variable has to be identical with the path name indicated with `./configure --prefix` for the source code distribution (i.e. it points to the GDMP install directory). GDMP client applications require this path.
- `GDMP_LOCAL_HOST=host1.cern.ch:` (C)- local host name where GDMP is installed.
- `GDMP_LOCAL_DOMAIN=cern.ch:` (O) - domain name of GDMP host.
- `GDMP_PORT_NUMBER=2000:` (C) - Port number on which the GDMP server is listening.
- `GLOBUS_LOCATION=/opt/globus:` (C) - Globus installation directory
- `OBJECTIVITY_DIR=/usr/local/bin:` (O) - Objectivity installation directory - only required if Objectivity database files are replicated
- `ORBACUS_DIR=/usr/local/bin:` - (O) - Orbacus installation directory - only required if MSS interface to HRM is used, see Section 7.
- `GDMP_FLATFILE_ROOT_DIR=/pool/data/flatfiles:` (C) - A common directory path for all physical files has to be provided. All physical filenames then *have* to contain this path in their path names.
- `GDMP_STAGE_FROM_MSS=/usr/local/grid/gdmp/utils/stage_from_mss:` (O) - used for staging a file from a mass storage system to a disk pool. Refer to Section 7 for details.
- `GDMP_STAGE_TO_MSS=/usr/local/grid/gdmp/utils/stage_to_mss:` (O) - used for staging a file from a disk pool to a mass storage system. Refer to Section 7 for details.
- `GDMP_FILE_CATALOG_SCRIPT:` (O) create a listing of files in a directory. Refer to Appendix B for details.
- `GDMP_REP_CAT_MANAGER_DN=cn=RManager, dc=host2, dc=cern, dc=ch:` (C/O) these are compulsory parameters if a the Globus replica catalogue based on LDAP [13] is used. The parameters are specific to the LDAP setup and need to be checked with the administrator of the replica catalogue. It is only needed when publishing files to the LDAP replica catalogue.
- `GDMP_REP_CAT_MANAGER_PWD=secret:` (C) - this password is required if new information needs to be inserted into the LDAP replica catalogue. For search operations, the password is not required.
- `GDMP_REP_CAT_FLATFILE_COLL_URL=ldap://host2.cern.ch:2010/`
`lc=flatfiles, rc=replica-catalogue, dc=host2, dc=cern, dc=ch:` (C) - flat files and Objectivity files are separated in the replica catalogue. All flat files are in one collection whereas all Objectivity files are in another collection. A collection needs to be identified by a name which is then used in the Globus replica catalogue.
- `GDMP_REP_CAT_OBJECTIVITY_COLL_URL=ldap://host2.cern.ch:2010/`
`lc=objyfiles, rc=replica-catalogue, dc=host2, dc=cern, dc=ch:` (C) - flat files and Objectivity files are separated in the replica catalogue. All flat files are in one collection whereas all Objectivity files are in another collection. A collection needs to be identified by a name which is then used in the Globus replica catalogue.

- **GDMP_REP_CAT_URL=ldap://host2.cern.ch:2010/**
rc=replica-catalogue, dc=host2, dc=cern, dc=ch: (C) - every LDAP server is identified by a unique name and a port number
- **GDMP_NOTIFICATION_FOR_REPLICATE_GET=/usr/local/grid/gdmp/utils/notifcation_get:**
(O) when a remote site has successfully transferred a file from the local site, the local server is notified and the stated script is called (see Section 5.5).
- **GDMP_NOTIFICATION_FOR_PUBLISH_CATALOGUE=**
/usr/local/grid/gdmp/utils/notifcation_publish: (O) - a notification script is called when the local site publishes the export catalogue (see Section 5.5).
- **GDMP_OBJY_ROOT_DIR=/pool/objy:** (C) - A common directory path for all physical Objectivity files has to be provided. All Objectivity filenames then *have* to contain this path in their path names.
- **00_FD_BOOT/pool/objy/example_federation.boot:** (O) - boot file path for Objectivity federation.
- **GDMP_DEFAULT_NEW_FIDID=12345:** (O) - if a new federation is created by GDMP, it will assign the following federation ID. By default it is assumed that the federation exists already.

By default, the file **gdmp.conf** is supposed to be in **/etc**, **/usr/local/etc** or **/opt/edg/etc** but can be put to any user defined location. If a user defined location is used, the environment variable **GDMP_CONFIG_FILE** has to be set and point to the location of the file. Note that this is the only environment variable that needs to be set in order to run GDMP client applications. Next, GDMP can be configured automatically with **utils/configure_gdmp**:

```
configure_gdmp <gdmp-install-dir> <userid> <port>
```

The command **configure_gdmp** is executed as *root* and will properly configure the GDMP server in the system files (**/etc/services** and **/etc/inetd**). The script requires three input parameters: **GDMP_INSTALL_DIR** **userid** and **port**. The GDMP server will run under the user-ID **userid** and on the specified port.

In detail, **configure_gdmp** updates the **inetd** configuration file for the GDMP server. **Inetd** will accept incoming requests on a certain port and then calls the GDMP server to handle the request. Note that by default GDMP will be assigned a service called “gdmp-server”. If the configuration file finds already a GDMP installation on the same machine, the new server will be assigned the service name “gdmp-server-date” where “date” is the current date when the script is run. Thus, several GDMP servers can be configured on the same machine. If the file **/etc/services** is edited manually, please make sure that the GDMP server name needs to be “gdmp-server-*” where “*” can be any string. 3.4.

The file **GDMP_INSTALL_DIR/utils/gdmp_server_start** handles these interactions and is configured automatically by the installation program. Note that the script **gdmp_server_start** is the main script for starting the GDMP server and needs to be modified for the specific installation. For further information on **inetd**, an example configuration and possible problem shooting refer to Appendix A.

3.5 The GDMP Directories

When GDMP is installed correctly, the following directories are available in the GDMP installation tree:

```
bin  etc  doc  include  lib  tmp  utils  var
```

The directory **bin** contains all the GDMP client applications as well as the GDMP server. **doc** contains the complete documentation of GDMP. **var** contains the log files created during program execution. The stdout and stderr of the GDMP server and all clients are also redirected to files in this directory.

etc contains the certificate and key to be used by the GDMP server to authenticate itself to the other Grid nodes, and the **host_list** file containing information about all the subscribed remote hosts. Furthermore, it contains the **import_catalogue** file containing information about all the files which are to be transferred from the remote hosts and the **export_catalogue** file containing information about the new files on the local host that must be notified to the subscribers. Finally, also the **local_file_catalogue** is stored here.

tmp contains temporary files maintained by the server or different clients. **include** contains the Replica Catalogue user interface (in subdirectory ReplicaCatalog) as well as internal GDMP header files (in subdirectory gdmf). Finally, libraries for the Replica Catalogue and GDMP are stored in **lib**.

In the current version it is assumed that at one site where GDMP is installed, the server and the client applications are always used by the same user since the read/write permissions to the directories and files are set to a single user only.

3.6 GDMP Server Certificate

3.6.1 CERN Certificate

The GDMP server currently requires a special “service” certificate for authentication and authorisation. The default certificate is signed by CERN and thus machines running the GDMP server need to accept CERN signed certificates. For more details on CERN certificates and how to accept them refer to:

<http://globus.home.cern.ch/globus/ca/>

3.7 Globus Certificate

The CERN signed certificate is required within the DataGrid project. In order to support also the needs of the e.g. PPDG project, a Globus signed server certificate is available too and one needs to switch to the Globus certificate like follows:

```
gdmf.cert_update -switch
```

For more details on the script refer to end of Section 6.

4 Data Replication and Background for GDMP Usage

GDMP is a file replication tool for replicating read-only files of any data format. In addition to transferring files from one site to another and notifying about new files created locally, the Globus replica catalogue is used to store file information about all sites in a Virtual Organisation and having GDMP installed. For details on replica catalogues and file replication refer to the DataGrid WP2 design document [7].

Although any file format can be used, GDMP has a particular plug-in for Objectivity files where files are “attached” to an Objectivity federation². Most of the command line tools are by default configured for flat files rather than Objectivity files.

4.1 Directories and File Replication

GDMP manages files stored in a particular *storage root directory* on a local disk or mounted disk pool. It requires all physical files to be stored in this directory structure which can be configured for each GDMP installation and thus for each storage system. Note that we distinguish between “flat files”³ (this can be any arbitrary file format like ROOT, ZEBRA, etc.) and Objectivity files which require particular replication steps. GDMP also requires two different storage root directories for these two file formats.

We start with an example. We assume that a large disk pool is mounted on a host `host1.cern.ch`. Files are stored in the directory `/data/run1/`. In the directory `run1` several subdirectories can exist and a possible directory layout is as follows:

```
/data/run1/day1/file1
/data/run1/day1/file2
/data/run1/day1/file3
/data/run1/day2/fileA
/data/run1/day2/fileB
```

GDMP requires a common root path for the directory structure since it manages several files in the replication process. In our example the common directory and thus the storage root directory for flat files is `/data/run1`. GDMP uses a configuration variable called `GDMP_FLATFILE_ROOT_DIR` which needs to be set in the file `gdmp.conf` (see Section 3).

For Objectivity files a similar directory structure is required and the variable `GDMP_OBJY_ROOT_DIR` must point to the storage root directory for Objectivity files.

The main task of GDMP is to mirror the directory structure of one storage system (called *Storage Element* in DataGrid terminology) to another. Thus, a storage root directory is required on both sites that participate in the mirroring process. Note that the storage root directory does *not* have to be identical on all Storage Elements but can be chosen and configured based on the local directory structure. If we now assume that all the files above need to be replicated to a Storage Element at Fermilab, the destination host first needs to set up a storage root directory. For example, on `host1.fnal.gov` (at Fermilab), `GDMP_FLATFILE_ROOT_DIR` has the value `/largedisk/cms/production/run1`. The GDMP replication process now can replicate files from the storage root directory on the source machine to the one on the destination machine.

4.2 Filenames

When files are replicated, identical files (replicas) exist at multiple locations and need to be identified uniquely. A set of identical replicas is assigned a *logical filename* (LFN) and each single physical file is assigned a *physical filename* (PFN). In [7] we also include a *transfer filename* (TFN) but we do not discuss it further and we will assume that all PFNs have the

²We do not discuss the details of Objectivity but point out that for Objectivity database files additional replication steps are necessary, as indicated when GDMP command line tools are explained.

³Note that the term flat file should *not* be regarded as offensive but rather to distinguish these files from files that need certain pre- and post processing steps as it is the case for database files.

complete paths used by the Storage Element's file system to refer to files resident on disk. The PFN also contains the host name i.e. the domain name of the Storage Element (see "Section 4.2 File Replication", in [7]) where the file is located and accessible via a Grid file transfer tool such as a GSI-enabled FTP server.

A typical example of a physical filename under the above assumptions is as follows:

```
pfn://host1.cern.ch/data/run1/day1/file1
```

We observe that when PFNs are used with GDMP, the prefix "pfn://" is not required. Once the file is created and the PFN is available, it can be inserted to the replica catalogue that provides a global name space for file replicas. See "Section 5.1.1 Replica Catalog" in [7] for details on replica catalogue issues. Note that for this GDMP version we use a single replica catalogue located at a single host (rather than a distributed replica catalogue as presented in [7]). In addition to the PFN, a logical filename must be assigned to the physical file. In the current version of the Globus replica catalogue, the LFN is equal to the last component of the PFN, i.e., to the file name including parts of the directory structure. This is a current restriction that will be removed in future versions. Thus, the logical filename is created automatically by GDMP from the physical filename complying with the implicit mapping enforced by the replica catalogue and for the physical filename in the example above it looks as follows:

```
day1/file1
```

4.3 Globus Replica Catalogue

In this subsection we describe how the Globus replica catalogue needs to be set up to interact with GDMP.

GDMP uses the Globus replica catalogue implementation which is based on LDAP. For details on the Globus replica catalogue refer to the user guide in [5]. The replica catalogue service is based on the LDAP protocol and a database backend where all replica information is stored. In principle, any possible LDAP server and a corresponding database backend can be used. The currently adopted solution is to use the OpenLDAP server and one of the database backends supported by OpenLDAP. For Solaris and Linux platforms we have tested the SleepyCat database backend (Sleepycat Berkeley DB 2.7.7: (08/20/99)). For Linux we tested the OpenLDAP database backend ldbm. Details on LDAP configuration can be found in [4].

As pointed out in Section 3, the following three LDAP replica catalogue variables have to be set in `gdmp.conf`:

```
GDMP_REP_CAT_MANAGER_DN
GDMP_REP_CAT_MANAGER_PWD
GDMP_REP_CAT_URL
```

Since we distinguish between flat files and Objectivity files as regards storage root directories, we need to do the same for replica catalogue configuration. The Globus replica catalogue provides the concept of *collections* that group several logically related files. Collections are not used explicitly in the GDMP user interface, but one collection for flat files and another collection for Objectivity files are implicitly created by GDMP in the replica catalogue to manage the two types of files. Thus, the following two configuration variables have to be set in

`gdmp.conf` to specify the two collections:

- `GDMP_REP_CAT_FLATFILE_COLL_URL`: collection of all flat files
- `GDMP_REP_CAT_OBJECTIVITY_COLL_URL`: collection of all Objectivity files

Note that all sites inserting information in the replica catalogue of a single Virtual Organisation, such as an experiment in a HEP environment, need to use the same collection URL.

4.4 Internal File Catalogues

GDMP uses a few catalogues that are used for internal book keeping and monitoring of the replication process. Once a site has finished writing a set of files (or just a single file), every single file needs to be registered in a *local file catalogue* which only contains files that are available at the local site. This catalogue contains the physical filename and logical file attributes like logical filename, file size, creation time, CRC checksum, file type. The local file catalogue is hidden from outside users and is thus only visible to the local GDMP server. The client application `gdmp_register_local_file` is used for inserting files to this catalogue.

At a certain point in time, a site can decide to publish its local files to other Grid sites using `gdmp_publish_catalogue`. In detail, all file entries of the local file catalogue are written into the replica catalogue⁴ and also sent to subscribed sites at remote sites (see Section 5.4). A list of all newly published files and their related information is written to a local *export catalogue*. The consumer site that wants to receive files creates an *import catalogue* where it lists all the files that are published by the producer and have not yet been transferred to the consumer site. The import catalogue holds the host name of the FTP server and all related physical and logical file information for each file. Figure 1 illustrates this model graphically.

When files are published, all required file information is read from the local file catalogue. Since the catalogue holds file attributes like size, during the execution of the publish command the files to be published do not need to reside on their physical location on disk but can already have been staged to a mass storage system. Note that files have to be at the disk location when `gdmp_register_local_file` is called since file size and CRC check sum are automatically created by GDMP and then stored in the local file catalogue.

5 Using GDMP

In this section we provide information on using the GDMP server and client applications and refer to Section 6 for detailed parameters for the command line tools.

5.1 Quick Start Guide to run GDMP

Follow these five steps to run the GDMP server and transfer files quickly and securely.

Assume that files are to be transferred from site A to B.

1. Register `gdmp_server` as an inetd service on site A as user 'x' and site B as user 'y'. This is done automatically by the installation program.

There are no restrictions on 'x' and 'y'.

⁴An insertion to the replica catalogue can also be disabled.

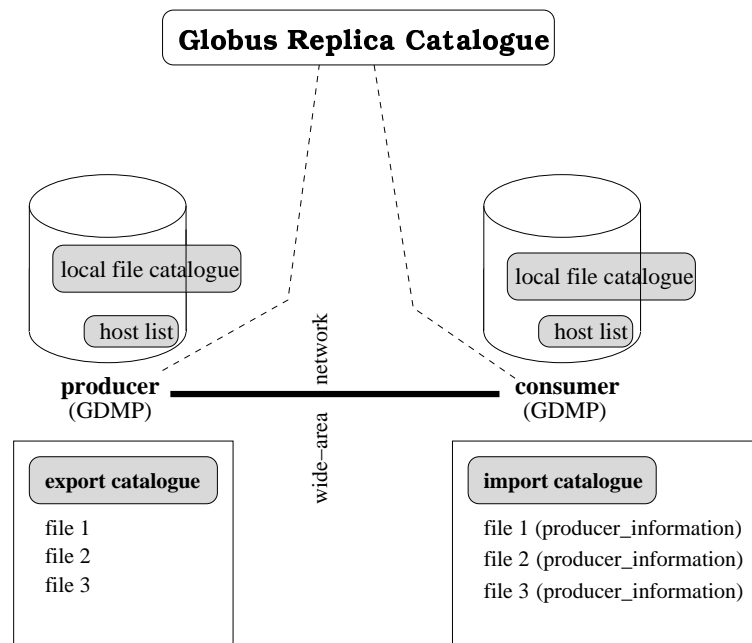


Figure 1: The role of the local file, export, import catalogues

You can specify the grid-mapfile to be used by the server using the ‘-m’ option. The default is `/opt/globus/etc/grid-mapfile`. You can make your own grid-mapfile on the same format as used by Globus and it will work but make sure that it has correct file access permissions.

Result: The server should start. The stderr and stdout are redirected the files `GDMP_INSTALL_DIR/var/gdmp_server.out` and `gdmp_server.err`.

2. Run `gdmp_host_subscribe` on site B as a user ‘y’, giving the host and port of site A.

You should first get a proxy using `grid-proxy-init`. For more details on Grid security refer to Appendix C.

This user ‘y’ should be in the grid-mapfile used by the server at site A.

Result: This should add the information about host on site B in the `GDMP_INSTALL_DIR/etc/host_list` file at site A.

3. Run `gdmp_register_local_file` on site A as user ‘x’.

Files stored in the storage root directory (e.g. `/data/run1` in Section 4.1) are registered. In case of Objectivity files: the lock-server of the related bootfile should be running.

Result: The files in the corresponding directory are inserted into the local file catalogue.

4. Run `gdmp_publish_catalogue` on site A as user ‘x’.

You should get the proxy for the client using `grid-proxy-init` once.

Files can only be published if they are in the local file catalogue.

This user 'x' should be in the grid-mapfile used by the server at site B.

This user 'x' has to be mapped to a local user which has read/write access in `GDMP_INSTALL_DIR/tmp` and `GDMP_INSTALL_DIR/etc` directories on site B. This mapping is required in the grid-mapfile used by the FTP server which by default is in `"/etc/grid-security/grid-mapfile"`.

Result: This should create a file `GDMP_INSTALL_DIR/etc/import_catalogue` on site B which will have the details of files at site A. Replica information by default is inserted to the replica catalogue.

5. Run `gdmp_replicate_get` on site B.

In case of Objectivity files, the lock-server of the `OO_FD_BOOT` should be running.

The `GDMP_STORAGE_ROOT_DIR` should have enough space to hold these files.

Result: This should transfer the files from site A to site B and validate them. In case of Objectivity files, the files are attached to the federation at site B. Replica information by default is inserted to the replica catalogue.

5.2 The Mechanics of GDMP

Registering a file in the Local File Catalogue

Once files are available on disk for replication to remote sites, the files need to be registered in a local file catalogue. This catalogue keeps track of all files that GDMP manages. GDMP mirrors *file sets* (a set can also contain just a single file) and automatically detects which files have been added to the local catalogue but have not been published yet. By inserting files into the local catalogue, GDMP gets control over the files and the corresponding replica information. Consequently, every single file needs to be registered first before it can be replicated. `gdmp_register_local_file` needs to be used for this. A possible *local file catalogue* looks like follows (in one single line):

```
flatfile:host1.cern.ch_.pool.data.testfiles:file1:1012039680:
603077316:1001660649
```

Which corresponds to:

```
filetype:file_id:relative_file_path_under_the_root_dir:size:checksum:timestamp
```

`filetype` can either be "flatfile" or "objectivity". Note that `file_ID` corresponds to a unique logical filename which is currently maintained by GDMP and cannot be changed. `checksum` is created by GDMP using the checksum command line tool. `timestamp` is a integer value for a file modification time.

Creating the Export Catalogue

The export catalogue contains information about all the files which are ready to be exported to other sites. The program `gdmp_publish_catalogue` is the trigger for the replication mechanism. This program has to be called when new files are ready for transportation to another site. It will compare the current and old `local_file_catalogue`. New file entries are written in the file `GDMP_INSTALL_DIR/etc/export_catalogue` which

lists all the new files that have been added to the local file catalogue since the last time `gdmp_publish_catalogue` was called.

A possible *export catalogue* looks like follows:

```
flatfile:host1.cern.ch_.pool.data.testfiles.file1:file1
```

Which corresponds to:

```
filetype:file_ID:filename
```

`file_ID` corresponds to a unique logical filename which is currently maintained by GDMP and cannot be changed. `filename` is the relative path starting from the directory after `storage_root_dir` which is either `GDMP_FLATFILE_ROOT_DIR` or `GDMP_OBJY_ROOT_DIR` (depending on the file type).

Publishing the Export Catalogue

The file update is based on the following fact: newly added files to the local file catalogue are detected and a difference between the new and the old local file catalogue is created. Based on the differences with the old catalogue, the export catalogue is created and transferred to all hosts subscribed and listed in the file `GDMP_INSTALL_DIR/host_list`. A client can only send this host list to a remote server if the user running the client is present in the grid-mapfile(s) being used by the FTP server and the `gdmp_server` on the remote machine. The export catalogue is renamed to `import_catalogue` at the destination site in order to distinguish between imported export catalogues and locally created export catalogues. Thus, the import catalogue holds the list of files which have been created newly by a remote site.

A possible *import catalogue* looks like follows:

```
flatfile:host1.cern.ch_.pool.data.testfiles.file1:file1:
/pool/data/testfiles:host1.cern.ch:3001:1
```

Which corresponds to:

```
filetype:file_ID:filename:storage_root_dir:hostname:port:flag
```

`file_ID` corresponds to a unique logical filename which is currently maintained by GDMP and cannot be changed. `filename` is the relative path starting from the directory after `storage_root_dir` which is either `GDMP_FLATFILE_ROOT_DIR` or `GDMP_OBJY_ROOT_DIR` (depending on the file type). `hostname` and `port` are the host name and port of the remote host where the file is located. In general, all information in the import catalogue refers to the remote file where it resides physically. `flag` is an internal flag to GDMP which is used when a remote file needs to be staged and is not registered in the replica catalogue.

Transferring the files

The remote site can decide when to start the data transfer from the remote to the local site. The program `gdmp_replicate_get` uses an FTP client library to securely transfer files. The program transfers each of the files listed in the import catalogue automatically to the local site. Each file is validated on arrival using the CRC checks, (is attached via `ooattachdb` in case of an Objectivity file to the local federation) and the file entry is deleted from the import catalogue. Finally, a file is registered in the replica catalogue.

5.3 Security Issues for GDMP Server and Clients

A GDMP server must run at each data production site. The port generally used is 2000, however you can set any other port through the inetd by editing the `/etc/services` file. Note that you will need root privileges to do this.

The server flags `-m` and `-l` can be used to specify the grid-mapfile and a Grid log file, respectively.

Note that the server uses its own service certificate and creates its Grid proxy automatically. Furthermore, the proxy is acquired for an unlimited time while client proxies are created by the user and by default are restricted to 12 hours. For large data transfers, the client proxy might not be available long enough to transfer several Gigabytes. Thus, it is recommended that the transfer time for a set of files be estimated and the proxy time be adjusted accordingly with the `-hours` option:

```
grid-proxy-init -hours xxx
```

where xxx corresponds to the number of hours the proxy will be available.

5.4 Subscription to Remote Servers

The Data Grid is most efficient when many hosts are part of the whole Grid and hence data is available in many different sites. Since a new site has to announce that it is available in the Grid and ready to get notified about the creation of files and replicas, the program `gdmp_host_subscribe` is used to subscribe the local host to any remote host. The local host will then be integrated into the host list of the remote host. When a remote host has written new files, the notification message and the export catalogue are transferred to each host in the host list. Currently, a local host has to subscribe to each remote host separately.

5.5 Notification

When a user publishes a local file catalogue with `gdmp_publish_catalogue`, a remote server gets notified and calls a configurable script which can then be used by external programs to start a data transfer request. In principle, `gdmp_replicate_get` can be executed and a fully automatic replication process can be set up. On the other hand, a similar notification script is called at the producer site when a consumer has successfully replicated a file. Thus, producers can keep track of consumers requesting and replicating files and can delete files again if local storage space is required. The following two variables in `gdmp.conf` need to point to the notification scripts:

```
GDMP_NOTIFICATION_FOR_REPLICATE_GET  
GDMP_NOTIFICATION_FOR_PUBLISH_CATALOGUE
```

As for the replicate get notification, GDMP will pass 2 arguments to this script: 1) full file name 2) host which has transferred the file. A possible example is as follows:

```
GDMP_NOTIFICATION_FOR_REPLICATE_GET /root/dir1/file1 host1.fnal.gov
```

For the publish notification, three arguments are passed to the script: 1) host name which has published files 2) file type 3) filename which contains the list of all newly published files. It has the same format as import catalogue, and so one can pass this to `gdmp_replicate_get` with the option `-c`. For example, a set of flat files has been published:

```
GDMP_NOTIFICATION_FOR_PUBLISH_CATALOGUE host1.fnal.gov flatfile /some/dir/catalogue
```

5.6 System States for File Replication Process

In the entire replication process, we distinguish several system states that are indicated by files with particular extensions. Thus, one can check the current status of a file transfer and discover possible problems. All the system states are stored in the directory `var` in the GDMP installation tree.

We now assume that a file “largeFile.extension” is transferred. In particular, a client at site A requests the file from site B using `gdmf_replicate_get`. The file name is first retrieved from the import catalogue. The **file ID** (stored in the local file catalogue) is used for indicating files and a particular extension (see below) is added to a status file of 0 size in the directory `var`.

The file ID contains the host name and all path information including the actual file name. In principle, the file ID corresponds to a logical filename and is unique for all replicas of this file. A file that should be transferred from `host1.cern.ch/data/file1` has the following file ID:

`host1.cern.ch_.data.file1`

After the host name a “_” is attached and “/” is converted to “.” The following transfer states are defined:

- **file_ID.stat**: the transfer of a file is currently going on or the file has been requested for transfer.
- **file_ID.replicated**: a file has been copied to its destination but not yet validated nor registered.
- **file_ID.validated**: a file has been validated locally after the copy process. The validation contains file size checks and CRC checksum comparison with the original file at the remote site.
- **file_ID.registered**: a file has been successfully validated and registered to the local file catalogue.
- **file_ID.transferred**: the file has already been successfully transferred, validated and registered but not yet deleted from the import catalogue. The tool `gdmf_catalogue_cleanup` needs to be used to delete these files plus the entries in the import catalogue. Once the file is in “transferred state”, it can be used by applications.
- **file_ID.notified**: a remote file has been notified about the correct file transfer (including validation and registration).
- **file_ID.req**: a file is requested from a remote site and is on tape rather than on disk. A staging request is initiated (see Section 7 for details on mass storage systems).
- **file_ID.done**: This file only exists for a very short time: when a staging requests has been done at a remote site (from MSS to disk) and the local server can start to get a file. Once the file transfer starts, the file is renamed to **.stat**.

In addition to the successful states, we also have a few error states:

- **file_ID.not_validated**: file validation failed
- **file_ID.not_registered**: file is not registered to the local file catalogue
- **file_ID.not_notified**: file notification failed
- **file_ID.error**: an error has occurred during the attachment of Objectivity files.

5.7 Replication Policy

The current replication policy available in GDMP is the following: a local site notifies other remote sites if new files are available. The remote sites in turn can decide when to transfer data. Currently, it is not possible for a site to actively transfer files to a remote site because of security reasons (put files rather than get files). However, such a replication policy can be integrated into a future software release if required.

5.8 Network Failures

In principle, each site that has not been reachable for some time is responsible for getting the latest information from other sites. Once a site is up and the network is working properly, the file `gdmp_get_catalogue` can be used to get the latest information from any site.

When a file transfer fails because of a network problem during the data transfer process, this transfer can be resumed afterwards when the network is back again. Thus, the data transfer continues from the latest checkpoint and prevents re-sending the whole file. This is based on the “resume” feature of the nc-ftp client.

5.9 Some Program Restrictions

The server should be run on a machine which is accessible from the outside world.

In the current version, all clients need access to the `GDMP_INSTALL_DIR/tmp` and `GDMP_INSTALL_DIR/etc` directories on the server side. So the local user, to which the client user is mapped in the grid-mapfile used by the FTP server, should have read/write privileges in these directories. This will be solved in later versions.

The Objectivity “bootfilepath” given to the configuration file should be *exactly* the same as specified in the Objectivity’s catalogue. Note that alias names for directories cannot be used.

6 GDMP Tools

In this section we describe the command line tools for GDMP and give details on the interaction with remote servers and the replica catalogue.

`gdmp_catalogue_cleanup`:

This tool provides a crash recovery functionality in cases when a remote site using a Mass Storage System has crashed. In addition, the import catalogue is cleaned up for all files that have been transferred successfully, i.e. file entries in the import catalogue are removed. The tool deletes all temporary files (files with the extensions `.req` and `.transferred` in the GDMP directory `var`) and recovers from crashes taking place during the remote staging of files. Note that this command is also called automatically after `gdmp_replicate_get` has succeeded transferring files. Refer to Section 5.6 for further information on status information about the file transfer.

Usage: `gdmp_catalogue_cleanup` :

`[-c <catalog>]` name of import catalogue

`[-t <flatfile|objectivity|all>]` type of catalog, default value is ‘all’

`[-h]` for help message

For further details on the use of a Mass Storage System with GDMP refer to Section 7.

gdmp_filter_catalogue: This functionality allows a partial-replication model where not all the available files in a local file catalogue are replicated. In detail, one or more filter criteria can be applied to the import and/or export catalogue in order to sieve out certain files. Refer to Section 5 for further details on import and export catalogues.

Two types of filters are supported: positive and negative filters. “positive” filter selects files that satisfy a certain filter criterion, keeps these files in the catalogue and deletes all other file entries whereas a “negative” filter deletes files from the catalogue which satisfy at least one criterion. Filters can be used also with `gdmp_replicate_get`, `gdmp_get_catalogue`, and `gdmp_publish_catalogue`.

An example for a positive filter is when a site only wants to publish files which have, e.g. the string “Rec” or “Tag” in the filename. As for a negative filter, a site may not want to replicate files containing the word “Calio” in the filename. Hence, the export catalogue, which contains all the files that a site wants to publish, has to be filtered so that file entries that match the filter criterion are deleted. Note that the original files are not deleted but only the file entries in the import or export catalogue. Furthermore, a site that does not want to get certain files from other sites can use a filter to specify which files it wants to have. In this case, the filter has to be applied to the import catalogue.

The `gdmp_filter_catalogue` command has an interactive interface with the following four menu items:

- add filter: several filter criteria can be created for the import or export catalogue. Note that separate filter criteria have to be created for the import and the export catalogue.
- read filter: Displays the filter criteria on the screen.
- delete criterion: One or more of the filter criteria can be deleted.
- apply positive filter: This function applies a positive filter on the catalogue. It is also possible to filter the catalogue by supplying a filter as a parameter to several of the GDMP tools (see below).
- apply negative filter
- help

The criteria are stored in the files `GDMP_INSTALL_DIR/etc/import_catalogue_filter` and

`GDMP_INSTALL_DIR/etc/export_catalogue_filter`. In order to start the program, one of the options has to be used:

```
Usage: gdmp_filter_catalogue :  
      (-i | -e)  
          -i for import catalogue  
          -e for export catalogue  
[-c <catalogue_path>] path of import/export catalogue.  
[-h ] for help message
```

Example: A site does not want to publish files that have the word “Jet” or “Muon” in the filename. These two criteria have to be added via the tool stated above. On publishing the export catalogue, this filter is applied on all the files listed in the export catalogue.

gdmp_get_catalogue: In case of a network or server failure, a certain host may not be notified when an update is done. A site which is down for some time is responsible for getting the latest information from other sites. This program contacts a certain host, gets the remote file catalogue and creates the corresponding import catalogue locally. A filter parameter can be applied optionally to filter the import catalogue.

```
Usage: gdmp_get_catalogue :
-r <remotehost> chooses a host from the host list
-p <remoteport> chooses a host port from the host list
[-t <flatfile|objectivity>] file type, default value is 'flatfile'
[-c <catalogue_path>] file in which you want to save the imported catalogue.
[-f <pos|neg>] filter the catalogue (pos or neg) after import.
[-h ] for help message
```

The host name has to be the name of a host which appears in the host list. The information about the remote port and directories which are necessary to get the catalogue is taken from the host list in `etc/host.list`. The local site should already have a local file catalogue although it is permitted to be empty.

GDMP creates two different catalogues for flat files and Objectivity files, respectively. The option `-t` is used to specify which kind of catalogue is required.

With the option `-c` the file catalogue of the remote site is created locally in the file to be specified as a second parameter without creating an import catalogue.

gdmp_host_subscribe: A host can subscribe to any other host in the Grid in order to be notified when updates of files are done.

```
Usage: gdmp_host_subscribe :
-r <remotehost> remote host name
-p <remoteport> remote port number
[-h ] for help message
```

The information about the local host is retrieved from the GDMP configuration file (`gdmp.conf`) and is sent to the specified remote host.

gdmp_ping: This tool checks if the GDMP server is running on a particular host and port and accepting clients. The remote server acknowledges this request and the message “The GDMP server is listening” is printed at the client side. The client uses a timeout value of n seconds (default is 10 seconds but can be changed with the timeout option `-t`). Within these n seconds, the server has to send an acknowledgement. If the server does not respond, the client terminates the connection to the server and assumes that the server is currently not ready to accept messages from the client. Thus, this tool should be called before a file transfer is started in order to ensure that the remote site is responding correctly.

```
Usage: gdmp_ping :
-r <remotehost> remote host name
-p <remoteport> remote port number
[-t <timeout>] timeout value in seconds
[-h ] for help message
```

gdmp_publish_catalogue: This tool must be used *after* `gdmp_register_local_file`. It creates the export catalogue locally (based on the content of the local file catalogue), sends a copy to the subscribed hosts and registers complete replica information (LFN, PFN, size, times tamp, CRC checksum, file type) into the replica catalogue. Note that the local host must have write access to the directory `GDMP_INSTALL_DIR/etc` of the remote host where the corresponding import catalogue is created.

```
Usage: gdmp_publish_catalogue :
[-t <flatfile|objectivity>] file type, default value is 'flatfile'
[-n ] do not update the Replica Catalogue
[-f <pos|neg>] filter the catalogue (pos or neg) before publish.
                It will only filter the newly add files and then publish.
[-h ] for help message
```

When publishing files, either flat files or Objectivity files can be published. If the option `-n` is used, replica information is not entered into the replica catalogue.

gdmp_register_local_file: GDMP keeps a local file catalogue for all files that it manages, thus every file that needs to be published and later replicated has to be registered in the local file catalogue (`GDMP_INSTALL_DIR/etc/local_file_catalogue`) by means of this command. Since each replica needs to be uniquely identified, a replica is assigned a logical filename (LFN) and a physical filename (PFN). The PFN is the actual file path on a disk location. The LFN is part of the PFN and currently is assigned by GDMP due to a restriction of the Globus replica catalogue. Refer to Section 4.2 for more details on file naming issues.

```
Usage: gdmp_register_local_file :
-p <pfn>|-d <directory>
                to add a single file use -p and
                to add all files with in a directory use -d
[-t <flatfile|objectivity>] specify the file type, default value is 'flatfile'
[-h] for help message
```

The tool stores the logical and physical filenames, the size, the modification time, the CRC checksum, and the file type in the local file file catalogue. Currently, the LFN cannot be chosen but in future versions it will be possible to assign a user defined logical filename to a set of identical replicas. One can register a single file (`-p`) with its entire physical filename (PFN) or all files of a directory (`-d`). In both cases, the logical filenames are assigned automatically based on the PFN and the storage root directory. Based on the file type (flat file or Objectivity files), additional information will be stored in the local catalogue.

Once the file is registered in the catalogue, it can possibly be staged to a mass storage system before `gdmp_publish_catalogue` is called. Information in the local file catalogue will be stored in the replica catalogue and then made available to the Grid. Note that this tool has to be used before `gdmp_publish_catalogue` is initiated!

gdmp_replicate_get: This is the main executable to transfer files from a remote machine to the local host. The users should make sure that they do not have an older file in the same directory and with the same name where the new file will be transferred to.

If no parameter is given, the default location of the import catalogue is used. By using the option `-c` a particular import catalogue can be specified but it has to be in the correct format. The import catalogue can be filtered for excluding particular files. The transfer results can be written into a log file.

It is possible to start multiple `gdmp_replicate_get` clients on the same import catalogue. The system itself takes care of concurrency issues and whether a file is already being transferred by some other client or not. This functionality provides users with an easy way to do parallel transfers and improve the network throughput obtained.

```
Usage: gdmp_replicate_get :
  [[-c <catalogue>]      import catalogue path |
   [-i <fileid>          LFN of the file
   -r <remotehost>       host from which file to transfer
   -p <remoteport>       port on which the remote gdmp server is listening
  ]]
  [-t <flatfile|objectivity>] filetype, default value is 'flatfile'
  [-d ] attach dummy DB instead of original(for Objectivity files only)
  [-n ] no update in Replica Catalogue
  [-f <pos|neg>] filter the import catalogue (pos or neg) first before
                  replicating files.
  [-h ] for help message
```

The default assumption is that files are taken from the import catalogue and transferred to the client machine. This allows a set of files rather than a single file to be replicated. However, if the file_ID of the file at the source site is known, the options `-i` can be used to replicate single files. A non-default import catalogue can be specified with the option `-c`.

By default, when a file is replicated successfully to a local site, the file is validated and then registered in the replica catalogue. In particular, the logical and physical filenames as well as size, modification time, CRC checksum and file type are registered in the replica catalogue. However, a site may request not to insert a file into the replica catalogue by using the option `-n`.

If the file type “objectivity” is chosen, replicated files are attached to a local federation. Since for large files the attach process can be time consuming, the option `-d` does a “dummy attach” (a simple trick where a different file is attached where Objectivity does not check for associations) that speeds up the attachment process.

Note that the data production site can store files in several directories. When these files are transferred to the local site, these directories are not also created in the local root

directory specified by `GDMP_FLATFILE_ROOT_DIR` or `GDMP_OBJY_ROOT_DIR`.

Some internal details:

The before `gdmp_replicate_get` starts a data transfer, it checks if the size of the requested file on disk corresponds to the file size in the remote `local_file_catalogue`. Only if the file sizes correspond, the file gets replicated. Thus, GDMP makes sure that the requested file is fully available at the remote site and not parts of the file only.

The tool also takes care of some error recovery before it starts to transfer files. It checks the status of transferred files and if it finds some `.not_registered`, `.not_notified`, `.not_replicated` or `.not_validated`, it will start the transfer process from this point again. For instance, if `.not_replicated` exists, it will start transferring it again. Furthermore, if `.not_validated` exists, this means that the file was not correctly transferred last time and it will start the transfer again.

gdmp_server: A server has to be started on each site participating in the Data Grid. The server is responsible for answering client messages, sending notification messages, and handling security issues. The server is started by the Internet daemon (`inetd`).

```
gdmp_server [Mapfile] [Logfile]
```

```
server options
```

```
-m          default value is /etc/grid-security/grid-mapfile
-l          Grid logfile, default: /var/gdmp.log
```

The port of the server is decided by the Internet daemon as specified in `/etc/services`. A server must have read access to the grid-mapfile of the local Globus installation. In this file the information about authorised users is stored. Option `-m` specifies the path of the Grid-mapfile and `-l` a log-file.

gdmp_stage_complete: is used only by the script `GDMP_STAGE_FROM_MSS` in order to automatically initiate a file transfer when a file has been staged from tape to disk. For further information refer to Section 7.

```
Usage: gdmp_stage_complete :
```

```
-f filename
```

```
[-h] for help message
```

The GDMP directory `utils` contains some shell utilities. It currently includes:

gdmp_server_start: This script is called by the `inetd` to start the GDMP server. See Appendix A Section 8 for installation details.

gdmp_cert_update: This script can either be used to update the GDMP server certificate or to switch between a Globus and a CERN certificate for the GDMP server. Note that the script reads the `GDMP_INSTALL_DIR` from `gdmp.conf`. One can set the environment variable `GDMP_CONFIG_FILE` to the location of the file `gdmp.conf` (see Section 3.4).

`gdmp_cert_update [-switch]`

The script can be used in the following two ways:

- If the script is called without the parameter “switch”, the newest GDMP server certificate is down-loaded from the GDMP CVS repository and then updated. This is necessary when the GDMP server certificate expires: 30 January 2002 for the CERN certificate and 30 July 2002 for the Globus certificate.
- If the option `-switch` is used, one can either switch from a CERN to a Globus certificate or vice versa. In general, the CERN certificate is the default one. The script will get the latest certificate from the CVS repository and update the GDMP configuration properly.

get_progress_report: The executable `gdmp_replicate_get` uses the `.stat` files in the GDMP directory `var` directory for each file currently being transferred. These files contain progress information on the transfer and are updated every few seconds. Once the file has been completely transferred, these `.stat` files are deleted and the final transfer log goes in the file `GDMP_INSTALL_DIR/var/replicate.log`. If you want to find out the state of advancement of all the transfers currently in progress, you can run this script, and it will produce a file `GDMP_INSTALL_DIR/var/progress.log` which will contain the latest progress information.

A typical progress report (stored in the file `progress.log`) looks like follows:

```
filename - total size - bytes transferred - %age completed
/data/file1 - 100000 - 50000 - 50%
```

A typical entry in the file `replicate.log` looks like follows:

```
/data/file1 incoming from host: host1.cern.ch, bytes transferred 100000,
start: [ Wed Oct 17 21:44:52 2001 ], end: [ Wed Oct 17 21:47:12 2001 ]
```

It contains the file to be transferred (the local file path), the start time and the end time of the data transfer for the specific file.

gdmp_version: prints the version number of the current release.

Further scripts like for staging and notification can be added here.

7 Support for a Mass Storage System

7.1 Motivation

GDMP supports a simple interface for a Mass Storage System (MSS). Since we assume that a site may run out of disk space, this interface is responsible for staging files from disk to the MSS when the disk-pool is full and staging them back to disk when requested.

7.2 Flow of Control

Let us assume a case where a file is in the local file catalogue but because of storage-space restrictions it has been staged to the MSS. Now this file is requested by a remote site via `gdmf_replicate_get`, and the file cannot be found in the directory **GDMP_FLATFILE_ROOT_DIR** (or **GDMP_OBJY_ROOT_DIR** in case of Objectivity files). GDMP always looks into this directory first. If the file is not found there, the remote client will send a request to stage the file. The local GDMP server will receive this request and will start a staging script which is pointed to by **GDMP_STAGE_FROM_MSS**. If the script is started successfully, the local server will send a message back to the remote client saying that staging is now in process. The remote client will disconnect and move on to the next files in its import catalogue. When the file staging is complete, the script will call `gdmf_stage_complete` which will notify the remote server that the file has been staged and is ready to transfer. The remote server will start a new client using `gdmf_replicate_get` to transfer this file.

7.3 The Interface

The staging script itself is not provided by GDMP. We only provide a plug-in mechanism for staging scripts (to and from the MSS) since different sites may have different Mass Storage Systems and thus require specific procedures. The staging script should have the following command line interface:

```
GDMP_STAGE_FROM_MSS <relative_path_name> <root_directory_on_disk>
```

where **relative_path_name** is a file path on disk and **root_directory_on_disk** is the storage root directory on disk (either for flat files or Objectivity files).

For instance, a possible script could be invoked as follows:

```
GDMP_STAGE_FROM_MSS dir1/dir2/file1 /data/directory
```

where the second argument would be mapped to the following physical file path on disk:

```
/data/directory/dir1/dir2/file1
```

It is required that the MSS itself has its internal catalogue about all files listed. Thus, it needs to map this file information here to its internal file location.

It is required that the staging process is atomic. GDMP partly takes care of this internally. GDMP has an internal method that checks the remote file size on disk and compares it with the one registered in the local file catalogue. Only if both file sizes are the same, GDMP starts the data transfer process. Thus, GDMP never transfers files that are partially staged to disk.

When the staging is completed, the script is expected to call the tool `gdmf_stage_complete` which would notify the client that the file is ready to be transferred.

The second staging script for transfers from disk to MSS should have the following interface:

```
GDMP_STAGE_TO_MSS <relative_path_name> <root_directory_on_disk>
```

The script should also have knowledge about the directory inside the MSS where the file should be copied to.

7.4 Staging States

We define the following states for processing of files that reside on tape at the remote site. Let us consider an example of a staging operation. A client requests the file `filename1` from a remote site with `gdmp_replicate_get`. Locally, this request is logged and the file `filename1.stat` is created in the GDMP directory `var`. If the file is on the remote disk, the local client carries on with the transfer, and this status file contains the progress of the transfer. However, if the file is not present on the remote disk, the local client sends a request to stage the file at the remote end and produces a file `filename1.req` in `GDMP_INSTALL_DIR/var`. The remote server calls the staging script. The script starts the executable `gdmp_stage_complete` when the staging has completed. This notifies the server on the requesting site that staging has been completed on the remote end. The server then starts the `gdmp_replicate_get` which starts the file transfer of `filename1` and creates the file `filename1.stat` indicating the transfer status.

7.5 Interface to HRM

GDMP has a plug-in for the Hierarchical Storage Manager (HRM) [1] APIs, which provide a common interface to be used to access different Mass Storage Systems. The implementation of HRM is based on CORBA communication mechanisms. Some initial integration tests have been performed, with promising results.

The C++ plug-in and the CORBA IDL exist in the source directories under: `StagingPlugins` and `HRMIDL`. We do not provide a production version of this now and thus the code is not included into the installation tree. For details on the IDL refer to:

http://cdcvs.fnal.gov/cgi-bin/public-cvs/cvsweb-public.cgi/ppdg_idl/

8 Appendix A: Configuring GDMP with inetd

This appendix gives a short introduction to inetd and explains which system files are edited locally doing the GDMP installation process.

8.1 Inetd Background

Simply put, the inetd provides Internet service management for a networked computer. It listens on certain ports and calls other servers or daemons to service request. As regards GDMP, we register a certain port, e.g. port 2000, with the inetd daemon and when a GDMP client connects to the machine via a socket connection, the inetd daemon takes the request on port 2000, starts the GDMP server via the script `GDMP_INSTALL_DIR/utils/gdmp_server_start` and passes all the socket information to the GDMP server. The GDMP server in turn then handles the client request.

In more detail, the inetd daemon starts by default each time the system is started. When the daemon starts, it reads its configuration information from the configuration files `/etc/services` and `/etc/inetd.conf`. GDMP is thus regarded as a service that is started by inetd.

8.2 Configuration Steps done by GDMP Installation Program

All the following steps are done automatically but we state them in some details for possible problem detection. In detail, the installation program inserts information into the files `/etc/services` and `/etc/inetd.conf` to set up a service.

The following steps are done by root:

1. Edit `inetd.conf` to add in one line:
gdmp-server stream tcp nowait "username"
"GDMP_INSTALL_DIR"/utils/gdmp_server_start
The terms in " " have to be replaced by the respective fields and "username" needs to be the userID of the user running the GDMP server.
2. Edit the `/etc/services` file to add
gdmp-server "port"/tcp
again "port" is the port of your choice; we use 2000 mostly.
3. Send the HUP signal to the inetd server: Linux users can get the inetd process id (pid) from `/var/run/inetd.pid`. On Solaris one can get the pid by doing `ps -e | grep inetd`.
To send the HUP signal to inetd you need to do:

```
kill -HUP <inetd_pid>
```

In principle, `gdmp_server_start` contains the environment variables `PATH`, `LD_LIBRARY_PATH` plus additional variables for GDMP, Objectivity and Orbacus.

Since inetd does not see the `PATH` and `LD_LIBRARY_PATH`, you can set them here but it depends on the system. Just do an `echo PATH` and `echo LD_LIBRARY_PATH` and add whatever you get in this file. Make sure you do not have any "echo"s in this file for debugging or whatever as the stdout is connected to the socket when the server is started with inetd.

8.3 Example Configuration for inetd

An example for a `/etc/inetd.conf` file entry is given here. GDMP is assigned the service name `gdmp-server` and the `gdmp_server_start` script will be called when `inetd` receives a request on the port 2000. We assume now that is the default GDMP port. The port itself is configured in the file `/etc/services` (see below). As for all examples in this User Guide we assume that the GDMP installation directory is `/usr/local/grid/gdmp`. Note that the entire sequence has to be added in one single line and the server is started as user “username”:

```
gdmp-server stream tcp nowait username \  
/usr/local/grid/gdmp/utils/gdmp_server_start
```

In the file `/etc/services` the GDMP server port is assigned:

```
gdmp-server      2000/tcp          # GDMP port
```

9 Appendix B: Example for gdmf.conf

Here we print the entire contents of the file gdmf.conf with a possible example as already introduced in Section 3. The comments in the configuration shall provide more detailed understanding:

```
#####
#GDMF Installation Directory.(COMPULSORY)
#This variable should be set otherwise GDMF commands will not work properly.
#####
GDMF_INSTALL_DIR=/usr/local/grid/gdmf

#####
#GDMF Local Host Full Name.(COMPULSORY)
#####
GDMF_LOCAL_HOST=host1.cern.ch

#####
#GDMF Domain Name(Optional)
#If not provided then if GDMF_LOCAL_HOST is localhost.domain.name
#then GDMF_LOCAL_DOMAIN will be domain.name
#####
GDMF_LOCAL_DOMAIN=cern.ch

#####
#Port number on which the GDMF server is listening.(COMPULSORY)
#It should be the same as you have used in the /etc/inetd.conf file
#####
GDMF_PORT_NUMBER=2000

#####
# Globus installation directory.(COMPULSORY)
#####
GLOBUS_LOCATION=/opt/globus

#####
#Objectivity installation directory.(OPTIONAL)
#####
OBJECTIVITY_DIR=/usr/local/bin

#####
#Orbacus installation directory.(OPTIONAL)
#####
ORBACUS_DIR=/usr/local/bin

#####
#Local/NFS mounted directory for Flat Files.(COMPULSORY)
#####
```

GDMP_FLATFILE_ROOT_DIR=/pool/data/flatfiles

```
#####
#Script/Executable to stage a file from MSS(OPTIONAL)
#This script will be called by GDMP to stage a file from MSS
#GDMP will pass 2 args to this script
#1) file_relative_path
#2) root_directory_where_this_script_should_copy_the_file
#e.g. If GDMP calls this script like
# $GDMP_STAGE_FROM_MSS dir1/dir2/file1 /root/directory
#then this script should copy the "dir1/dir2/file1" from MSS
#into "/root/directory/dir1/dir2/file1"
#####
GDMP_STAGE_FROM_MSS=/usr/local/grid/gdmp/utils/stage_from_mss
```

```
#####
#Script/Executable to migrate a file to MSS(OPTIONAL)
#This script will be called by GDMP to migrate a file to MSS
#GDMP will pass 2 args to this script
#1) file_relative_path
#2) root_directory_where_this_file_exists_on_disk
#e.g. If GDMP calls this script like
# $GDMP_STAGE_TO_MSS dir1/dir2/file1 /root/directory
#then this script should copy the "/root/directory/dir1/dir2/file1"
#from disk and save it into MSS
#####
GDMP_STAGE_TO_MSS=/usr/local/grid/gdmp/utils/stage_to_mss
```

```
#####
#To create a listing of files in a directory(OPTIONAL)
#GDMP will call this script with 2 args
#1) directory_for_which_GDMP_needs_listing
#2) file_in_which_this_script_should_save_the_listing
#e.g. if GDMP calls this script as
#$GDMP_FILE_CATALOG_SCRIPT /root/directory /tmp/tmp_file_234
#Then this script should search in the "/root/directory" directory
#and create a list of all files in this directory and save it in
#/tmp/tmp_file_234 file.
#each line in "/tmp/tmp_file_234" should have this format
#<filetype>:<logical_name>:<file_path>
#where,
# <filetype> =      filetype e.g. for flat files it should be 'flatfile'
#                  for Objectivity files it could be 'objectivity'
#                  or you can use 'default' which mean select the one
#                  which is passed as command line arg to
#                  gdmp_register_local_file, so if
#                  gdmp_register_local_file is called for flatfile
#                  then default will be changed to 'flatfile' and if
```



```

#           gdmf_register_local_file is called for Objectivity
#           files then default will be changed to 'objectivity'
# <logical_name> = logical file name for the file. Each file should have
#           a unique logical file name. One can set this value
#           to 'automatic' which mean gdmf should create a logical
#           file name for it.
#           If GDMP is going to create logical file name then
#           for Flat Files it will be,
#           GDMP_LOCAL_HOST_<modified_file_full_path>
#           where <modified_file_full_path> = each '/'
#           in the file full path will be replaced to '.'
#           and each '.' will be replaced to '..'
#           for Objectivity files it will be,
#           DB_SYSNAME.DBID
# <file_path>   = File full path. e.g if "/root/directory" directory
#           has a file dir1/file1 then <file_path> will be
#           "/root/directory/dir1/file1"
#If you don't set this variable then
#$GDMP_INSTALL_DIR/utils/create_file_export_catalog will be used
#which will create listing of the form
#default:automatic:<file_full_path>
#####
GDMP_FILE_CATALOG_SCRIPT=

#####
#DN of Globus Replica Catalog Manager(COMPULSORY, if you want to
#search/update)
#Contact the Replica Catalog Administrator to get the value of this
#####
GDMP_REP_CAT_MANAGER_DN="cn=RCManager, dc=host2, dc=cern, dc=ch"

#####
#Passwrod for Globus Replica Catalog Manager(COMPULSORY, if you want to
#search/update)
#Contact the Replica Catalog Administrator to get the value of this
#####
GDMP_REP_CAT_MANAGER_PWD=secret

#####
#Collection URL in which Flat Files info will be saved(COMPULSORY,
#if you want to search/update)
#Contact the Replica Catalog Administrator to get the value of this
#####
GDMP_REP_CAT_FLATFILE_COLL_URL=ldap://host2.cern.ch:2010/
lc=flatfiles, rc=replica-catalogue, dc=host2, dc=cern, dc=ch

#####

```

```

#Globus Replica Catalog URL(COMPULSORY, if you want to search/update)
#Contact the Replica Catalog Administrator to get the value of this
#####
GDMP_REP_CAT_URL=ldap://host2.cern.ch:2010/
rc=replica-catalogue, dc=host2, dc=cern, dc=ch

#####
#Script/Executable for the File Transferred Notification(OPTIONAL)
#GDMP will run this script when a remote site has successfully
#transferred a file. GDMP will pass 2 args to this script
#1) full_file_name
#2) host_which_has_transferred_this_file
#$GDMP_NOTIFICATION_FOR_REPLICATE_GET /root/dir1/file1 shahzad.fnal.gov
#One can write this script to log this info, delete the local file etc
#If site A is transferring files from site B then this script will be
#called at site B
#####
GDMP_NOTIFICATION_FOR_REPLICATE_GET=/usr/local/grid/gdmp/utils/
notification_get

#####
#Script/Executable for the File Publish Notification(OPTIONAL)
#GDMP will run this script when a remote site published some files
#GDMP will pass 3 args to this script
#1) host_name_which_has_published_files
#2) type_of_files
#3) Filename which contains the list of all newly published files.
# It has the same format as import catalogue, so one can
# pass this to gdmp_replicate_get with the option -c
#$GDMP_NOTIFICATION_FOR_PUBLISH_CATALOGUE shahzad.fnal.gov flatfile \
# /some/dir/import_catalogue_for_newly_published_files
#One can write this script to automatically start the
#gdmp_replicate_get to transfer the published files
#If site A is publishing files to site B,C and D then this script will
#be called at site B,C and D
#####
GDMP_NOTIFICATION_FOR_PUBLISH_CATALOGUE=/usr/local/grid/gdmp/utils/
notification_publish

#####
#Objectivity Related Variables.
#These values will be used only if you have built GDMP with
#Objectivity Option and want to transfer Objectivity files, otherwise
#you don't have to set these variables
#####
#Local/NFS mounted directory for Objectivity DB files.(COMPULSORY)
#####

```

GDMP_OBJY_ROOT_DIR=/pool/objy

```
#####
#Boot file path for the Objectivity Federation.(COMPULSORY)
#####
OO_FD_BOOT=/pool/objy/example_federation.boot
#####
#FDID for the Objectivity Federation(Optional)
#If you don't have a local Objectivity federation and you want to
#build one when you first time run gdmf_replicate_get then you can set
#this variable to assign the FD ID for the new Objectivity Federatoion
#If this value is not set then the default value will be used which is "1"
#####
GDMP_DEFAULT_NEW_FDID=12345
#####
#Globus Replica Catalog(GRC) Collection URL for Objectivity.(COMPULSORY)
#If you want to update the GRC for Objectivity files then set this
#####
GDMP_REP_CAT_OBJECTIVITY_COLL_URL=ldap://host2.cern.ch:2010/
lc=objyfiles, rc=replica-catalogue, dc=host2, dc=cern, dc=ch
```

10 Appendix C: Usage of Grid Security Infrastructure

This is a short introduction into the usage of Grid security in Globus and should be sufficient for using GDMP. It is intended for people new to Globus. For exact details, please refer to the Security section at the Globus web page: <http://www.globus.org/Security/>

GDMP assumes that Globus is installed at the local machine and requires the authentication and authorisation method of Globus. In particular, the Globus command **grid-proxy-init**. Please check if this command is available on your system.

In order to use Globus, one needs to have a special certificate and a key which are required for the authentication procedure. For certificate requests and further details on certificates refer to the Testbed and Integration web page of DataGrid at: <http://marianne.in2p3.fr/> and check out the side bar “Certification Authorities”.

We now assume that you have requested such a certificate and the key. The files **usercert.pem** and **userkey.pem** by default are stored in:

```
user_home_directory/.globus
```

Please make sure that the two files have the correct access permissions:

```
-r----- 1 username      group          963 Mar  8 2001 userkey.pem
-r--r--r-- 1 username      group          3873 Mar  8 2001 usercert.pem
```

Once all the files are correctly in place, a “proxy” can be gained via **grid-proxy-init** which provides a single log on to several machines where you are registered as a Grid user. Only when the proxy is valid (it expires after some time) GDMP tools can be used. The lifetime of the proxy can be checked with the Globus command **grid-proxy-info** like follows:

```
host1>grid-proxy-info -all
subject   : /O=Grid/O=CERN/OU=cern.ch/CN=Firstname Surname/CN=proxy
issuer    : /O=Grid/O=CERN/OU=cern.ch/CN=Firstname Surname
type      : full
strength  : 512 bits
timeleft  : 25:59:59 (1.0 days)
```

The subject name “/O=Grid/O=CERN/OU=cern.ch/CN=Firstname Surname” uniquely identifies a Grid user. Before you want to access a remote machine, e.g. set up a GDMP transfer between **host1.infn.it** and **host1.cern.ch**, you need to request the remote administrator to enter your subject name to the machines “grid-mapfile” that holds all authenticated users. Note that this only needs to be done once before the GDMP installation process.

11 Appendix D: Replica Catalogue API

11.1 Description of the Programming Interface

The Replica Catalogue API is a generic C++ API to the Globus replica catalogue. It implements most of the methods for the Replica Catalog as stated in the Data Management Architecture document, “Section Replica Catalog” [7]. For background on replica catalogues, logical and physical filenames please refer to this [7].

The C++ replica catalogue class (to be found in the GDMP installation tree under `include/ReplicaCatalog/ReplicaCatalog.h` is defined as follows and allows for insertion, deletion and search of replica information:

```
class ReplicaCatalog {
public:
    ReplicaCatalog(      RC_Url      contact_string,
                        string      manager_dn,
                        string      manager_pw,
                        string      collection_url);
    ~ReplicaCatalog(void);

    vector<string> getPhysicalFileNames(string lfn);
    string getLogicalFileName(string pfn);
    RC_Result addLogicalFileName(string lfn);
    RC_Result addPhysicalFileName(string lfn, string pfn);
    RC_Result deleteLogicalFileName(string lfn);
    RC_Result deletePhysicalFileName(string lfn, string pfn);
    RC_Result addLogicalFileAttribute(string lfn, string attrnam, string attrval);
    RC_Result deleteLogicalFileAttribute(string lfn, string attrnam, string attrval)
    deque<GDMP_AttrVal_Pair> getLogicalFileAttributes(string lfn);

private:
    RC_Url      rc_url_;
    string      collection_url_;
    GDMP_Rep_Catalogue* rep_catalog_;
};
```

Note that the variables in the constructor need LDAP specific information and need to be checked with the replica catalogue administrator.

11.2 Usage Instructions

The replica catalogue API does not depend on GDMP and can be compiled and linked without the GDMP source code. In order to use the API in an application, a the ReplicaCatalogue library as well as a few Globus and LDAP libraries need to be linked to the application (see example makefile below). Note that the ReplicaCatalog library is available as a static and shared library. If the shared library is used, then the directory `lib` of the GDMP installation tree needs to be added to the `LD_LIBRARY_PATH`.

A test program for the replica catalogue can be found in the directory `test` in the GDMP source code tree which is also available at the CVS repository at the address below. Please

make sure that `gdmp install` is called before `rc_addtest` is executed. If not, you need to include “lib/.libs” to the library path.

<http://cdcvs.fnal.gov/cgi-bin/public-cvs/cvsweb-public.cgi/gdmp/>

A possible makefile looks like follows:

```
g++ rc_addtest.C -g -o rc_addtest -I$GLOBUS_LOCATION/include \  
-I../ReplicaCatalogue -I$GLOBUS_LOCATION/include/gcc32dbgpthr \  
-L../lib/.libs -L$GLOBUS_LOCATION/lib \  
-lReplicaCatalog -lglobus_replica_catalog_gcc32dbgpthr \  
-lglobus_common_gcc32dbgpthr -lldap_gcc32dbgpthr -llber_gcc32dbgpthr
```

12 Appendix E: BrokerInfo API

12.1 Description of the Programming Interface

The BrokerInfo C++ API provides the user with an interface to job information that comes from the Job Scheduler. The Scheduler selects a Computing Element to send a job for execution and writes info about the selected resources in the BrokerInfo file which is shipped together with the job. Reading the BrokerInfo file through this interface, the application may retrieve info on the Computing Element where the job is running, the close Storage Elements, etc. The API provides also an implementation for methods that will be provided by the WP2 Replica Manager interface in the future.

A more detailed description of the BrokerInfo library and its functionality can be found in the BrokerInfo document provided by WP1 [8].

The C++ BrokerInfo and ReplicaCatalogB classes (to be found in the GDMP installation tree under `noindent include/BrokerInfo/BrokerInfoB.h` and `include/BrokerInfo/ReplicaCatalogB.h`) are defined as follows:

```
class BrokerInfoEx {
};

class BrokerInfo {
public:

    ~BrokerInfo(void);
    static BrokerInfo* instance(void);
    BI_Result getCE(string& CE) const;
    BI_Result getDataAccessProtocol(vector<string>& DAPs) const;
    BI_Result getInputPFNs(vector<string>& PFNs) const;
    BI_Result getLFN2PFN(string LFN, vector<string>& PFNs) const;
    BI_Result getSEs(vector<string>& SEs) const;
    BI_Result getSEProto(string SE, vector<string>& SEProtos) const;
    BI_Result getSEPort(string SE, string SEProtocol, string& SEPort) const;
    BI_Result getCloseSEs(vector<string>& SEs) const;
    BI_Result getSEMOUNTPoint(string CloseSE, string& SEMOUNT) const;
    BI_Result getPFNs(vector<string>& PFNs) const;

private:

    BrokerInfo(void);
    BI_Result vSearch(const char* searchstr, vector<string>& retvect) const;
    BI_Result sSearch(const char* searcharg, const string searchstr,
        int& position) const;
    BI_Result svIndexBuild(const char* sarg, const string sstr,
        const string varg, vector<string>& retvect) const;
    void vBuild(const string buildstr, vector<string>& retvect) const;

    string BrokerInfoFile_;
    ifstream fbrokerinfo_;
```

```

    strstream mbrokerinfo_;
    ClassAd* ad_;
    static BrokerInfo* instance_;
}

class ReplicaCatalogBEx {
};

class ReplicaCatalogB {
public:

    ~ReplicaCatalogB(void);
    ReplicaCatalogB(void);
    vector<string> ReplicaCatalogB::getPhysicalFileName(string LFN);
    string ReplicaCatalogB::getBestPhysicalFileName(vector<string> PFN,
    vector<string> Protocols);
    string ReplicaCatalogB::getTransportFileName(string PFN, string Protocol);
    string ReplicaCatalogB::getPosixFileName(string TFN);
    BI_Result ReplicaCatalogB::getSelectedFile(string LFN, string Protocol,
    string TFN, string FileName);

private:

    BrokerInfo *brokerinfo_;

};

};

```

12.2 Usage Instructions

The BrokerInfo API does not depend on GDMP and can be compiled and linked without the GDMP source code. In order to use the API in an application, BrokerInfo library as well as the Condor ClassAd library need to be linked to the application (see example makefile below). Note that the BrokerInfo library is available as a shared and static library. If you use the shared library, the directory `lib` of the GDMP installation tree needs to be added to the `LD_LIBRARY_PATH`.

A test program for the BrokerInfo can be found in the directory `test` in the GDMP source code tree which is also available at the CVS repository at:

<http://cdcvs.fnal.gov/cgi-bin/public-cvs/cvsweb-public.cgi/gdmp/>

A possible makefile looks like follows:

```

g++ BrokerTester.C -g -o BrokerTester \
-I../BrokerInfo \
-I/usr/local/grid/ClassAd/include \

```



```
-L../lib/.libs \  
-L/usr/local/grid/ClassAd/lib \  
-lBrokerInfo -lclassads
```

Acknowledgements

The GDMP project (originally called Grid Data Management Pilot) was started in early 2000 as a pilot project by Heinz Stockinger and Asad Samar to evaluate the Globus toolkit, take useful features for a file replication system and produce a prototype to be evaluated in a real production environment.

The software development process is already well advanced and the project is now a collaboration between the European DataGrid (in particular the Data Management work package) and the Particle Physics Data Grid (PPDG). Thus, the GDMP team was increased and we got lots of constructive feedback from our colleagues in DataGrid and PPDG. In particular, we want to thank the WP2 team (the bigger subset that is not already in the GDMP team): Wolfgang Hosc hek, Peter Kunszt, Javier Jaen-Martinez, Ben Segal, Kurt Stockinger and Brian Tierney.

From the US side, we want to thank the Globus team: in particular Bill Allcock, John Bresnahan, Ann Chevernak, Ian Foster, Carl Kesselman, Darcy Quesnel and Mike Wilde (we definitely have forgotten a few names (sorry about that) but these are the people we have most contacts with). Thanks also to Miron Livny (Univ. of Wisconsin) for good discussions which led to several useful additions to GDMP.

Versions 1.0 until 1.2.2

We keep the acknowledgement of older GDMP versions too since the following people gave constructive feedback and discussion during the first project phase:

We want to thank Tony Wildish (Princeton University) for initial work on a replication tool written in Perl. This tool [12] and personal discussions have provided us with important input for the architecture of GDMP. Furthermore, thanks to Koen Holtman (Caltech) who is re-using and testing parts of our code and has pointed out some bugs in the software. Thank you also to Shahzad Muzaffar (Fermi National Lab.) for taking part in the transatlantic replication tests and fixing bugs in GDMP. Thanks to Flavia Donno (INFN) for very valuable input and providing an installation procedure for GDMP within the INFN Installation. Finally we want to thank Luciano Barone (INFN), Dominique Boutigny (IN2P3), Johannes Gutleber (CERN), Mehnaz Hafeez (CERN), Koen Holtman (Caltech), Wolfgang Hosc hek (CERN), Bob Jacobson (LBL), Werner Jank (CERN), Javier Jaen-Martinez (CERN), Veronique Lefebure (CERN), Harvey Newman (Caltech), Ben Segal (CERN), Arie Shoshani (LBL), and Kurt Stockinger (CERN) for their input and valuable discussions. We are also thankful to the Globus team for providing support and technical advice on various issues.

References

- [1] L. M. Bernardo, A. Shoshani, A. Sim, H. Nordberg. Access Coordination of Tertiary Storage for High Energy Physics Application, *17th IEEE Symposium on Mass Storage Systems and 8th NASA Goddard Conference on Mass Storage Systems and Technologies*, Maryland, USA, March 27-30, 2000.
- [2] European DataGrid Project: <http://www.eu-datagrid.org>
- [3] Data Management Work Package in EDG: <http://grid-data-management.web.cern.ch/grid-data-management>

- [4] Andrea Domenici, Notes on the Usage of an experimental Replica Catalog for the CERN DataGrid Testbed, 14 August 2001. <http://www.cern.ch/grid-data-management/docs/ldapuse.ps>
- [5] Globus Project: Getting Started with the Globus Replica Catalog, <http://www.globus.org/datagrid/deliverables/replicaGettingStarted.pdf>
- [6] Mehnaz Hafeez, Asad Samar, Heinz Stockinger. A DataGrid Prototype for Distributed Data Production in CMS, *VII International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT2000)*, October 2000.
- [7] Wolfgang Hoesche, Javier Jean-Martinez, Peter Kunszt, Ben Segal, Heinz Stockinger, Kurt Stockinger, Brian Tierney. Data Management (WP2) Architecture Report - Design, Requirements and Valuation Criteria, *DataGrid-02-D2.2-0103-1_2*, http://grid-data-management.web.cern.ch/grid-data-management/docs/DataGrid-02-D2.2-0103-1_2.pdf, Geneva, Sept 19, 2001.
- [8] Flavia Donno, Salvo Monforte, Francesco Prelz, Livio Salconi, Massimo Sgaravatto. The Resource Broker Info File, *DataGrid-01-NOT-0113* http://www.pd.infn.it/~sgaravat/Grid/datagrid-01-not-0113-1_2.pdf Pisa, Sept 28, 2001.
- [9] Particle Physics Data Grid project (PPDG): <http://www.ppdg.net>
- [10] Asad Samar, Heinz Stockinger. Grid Data Management Pilot (GDMP): A Tool for Wide Area Replication, *IASTED International Conference on Applied Informatics (AI2001)*, Innsbruck, Austria, February 19-22, 2001.
- [11] Heinz Stockinger, Asad Samar, Bill Allcock, Ian Foster, Koen Holtman, Brian Tierney. File and Object Replication in Data Grids, *10th IEEE International Symposium on High Performance and Distributed Computing (HPDC-10)*, San Francisco, California, August 7-9, 2001.
- [12] <http://www.cern.ch/wildish>
- [13] Wengyik Yeong, Tim Howes, Steve Kille. Lightweight Directory Access Protocol, Request For Comments (RFC) 1777, March 1995.